Document number 2000007

AV/C Changer Subunit 1.0

April 12, 2004

Sponsored by:

1394 Trade Association

Accepted for publication by

1394 Trade Association Board of Directors

Abstract

This document defines commands and descriptor structures for the Changer Subunit.

Keywords

IEEE 13994, Serial Bus, AV/C, Changer

1394 Trade Association Specification

1394 Trade Association Specifications are developed within Working Groups of the 1394 Trade Association, a non-profit industry association devoted to the promotion of and growth of the market for IEEE 1394-compliant products. Participants in Working Groups serve voluntarily and without compensation from the Trade Association. Most participants represent member organizations of the 1394 Trade Association. The specifications developed within the working groups represent a consensus of the expertise represented by the participants.

Use of a 1394 Trade Association Specification is wholly voluntary. The existence of a 1394 Trade Association Specification is not meant to imply that there are not other ways to produce, test, measure, purchase, market or provide other goods and services related to the scope of the 1394 Trade Association Specification. Furthermore, the viewpoint expressed at the time a specification is accepted and issued is subject to change brought about through developments in the state of the art and comments received from users of the specification. Users are cautioned to check to determine that they have the latest revision of any 1394 Trade Association Specification.

Comments for revision of 1394 Trade Association Specifications are welcome from any interested party, regardless of membership affiliation with the 1394 Trade Association. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally, questions may arise about the meaning of specifications in relationship to specific applications. When the need for interpretations is brought to the attention of the 1394 Trade Association, the Association will initiate action to prepare appropriate responses.

Comments on specifications and requests for interpretations should be addressed to:

Editor, 1394 Trade Association
1111 South Main Street, Suite 100
Grapevine, TX 76051
USA

Published by

1394 Trade Association
1111 South Main Street, Suite 100
Grapevine, TX 76051 USA

Printed in the United States of America

Contents

Tables

Figures

Annexes

Foreword (This foreword is not part of 1394 Trade Association Specification 2000007)

This specification defines the model, commands, and descriptors of the AV/C Changer Subunit.

There are three annexes in this specification, all of which are informative.

This specification was accepted by the Board of Directors of the 1394 Trade Association. Board of Directors acceptance of this specification does not necessarily imply that all board members voted for acceptance. At the time it accepted this specification, the 1394 Trade Association Board of Directors had the following members:

Eric Anderson, Apple, Chair
Angela Lee, Mitsubishi Digital Electronics America, Inc, Vice-Chair
Dave Thompson, Agere Systems, Secretary

Organization Represented	*Name of Representative*
Oxford Semiconductor	Jalil Oraee
Hewlett-Packard	Alan Berkema
Molex	Max Bassler
Staccato Communications	Jason Ellis
Mindready	Martin Lessard
Newnex Technologies	Sam Liu
Quantum Parametrics LLC	Richard Mourn
Firecomms Ltd	Declan O'Mahoney
Samsung	Jong-Wook Park
VividLogic	Shiva Patibanda
Microsoft Corporation	Steve Powers
Texas Instruments	Cecelia Smith
Panasonic/Matsushita	Hans van der Ven

The AV Working Group, which developed and reviewed this specification, had the following members:

Tsuyoshi Sawada, Chair
Takuya Nishimura, Secretary

Ram Balareaman
Hiroyuki Chaki
Junichi Fujimori
Burke Henehan
Hiroyuki Iitsuka
Prashant Kanher
Jong Won Kim
Pascal Lagrange
Yuji Nakura
Takeshi Okauchi
Yasutoshi Onishi
Jalil Oraee
Mike Overlin
Jongwook Park
Song Hee Park
Shiva Patibanda
Jehu Pineda
Shigeki Takahashi
Michael Johas Teener
Thomas Thaler
Tommy Shih
Colin Whitby-Strevens
Andy Yanowitz

Revision history

Revisions prior to 0.8 have been developed as working drafts within the AV Working Group.

Revision 0.8 (December 2003).

- (Editorial) Formatted to the current 1394 Trade Association template.
- (Technical) Examples added to Annex C.
- (Technical) Compliance Annex added according to latest 1394 Trade Association guidelines

AV/C Changer Subunit 1.0

1 Scope and purpose

1.1 Scope

This specification define the model, commands, and descriptors of the AV/C Changer Subunit.

1.2 Purpose

The purpose of this document is to define the AV/C Changer Subunit. The Changer Subunit provides a media-independent control model for devices that implement media changer functionality.

2 Normative references

2.1 Reference s

The specifications and standards named in this section contain provisions, which, through reference in this text, constitute provisions of this 1394 Trade Association Specification. At the time of publication, the editions indicated were valid. All specifications and standards are subject to revision; parties to agreements based on this 1394 Trade Association Specification are encouraged to investigate the possibility of applying the most recent editions of the specifications and standards indicated below.

2.2 Approved references

The following approved specifications and standards may be obtained from the organizations that control them.

[R1] IEEE Std 1394-1995, Standard for a High Performance Serial Bus

[R2] IEEE Std 1394a-2000, Standard for a High Performance Serial Bus—Amendment 1

[R3] IEEE Std 1394b-2002, Standard for a High Performance Serial Bus—Amendment 2

[R4] AV/C Digital Interface Command Set General Specification Version 4.1. TA Document 2001012.

[R5] AV/C Information Block Types Specification Version 1.0. TA Document 1999045.

[R6] AV/C Disc Subunit General Specification, Version 1.2. TA Document 2002001.

[R7] AV/C Disc Media Type Specification – CD-DA, Version 1.0. TA Document 1999002.

[R8] AV/C Disc Media Type Specification – MD-audio, Version 1.0. TA Document 1998014.

[R9] AV/C Disc Media Type Specification – SACD, Version 1.2. TA Document 2001016.

[R10] AV/C Disc Media Specification – DVD. TA Document 2000001.

Throughout this document, the term “IEEE 1394" shall be understood to refer to IEEE Std 1394-1995 as amended by IEEE Std 1394a-2000 and IEEE Std 1394b-2002.

2.3 References under development

At the time of publication, the following referenced specification was under development.

[R11] AV/C Descriptor Mechanism Specification Version 1.2. TA Document 2002013.

2.4 Reference acquisition

The references cited may be obtained from the organizations that control them:

1394 Trade Association, 1111 South Main Street, Suite 100, Grapevine, TX 76051 USA; (817) 410-5750 / (817) 410-5752 (FAX); http://www.1394ta.org/

Institute of Electrical and Electronic Engineers (IEEE), 445 Hoes Lane, PO Box 1331, Piscataway, NJ 08855-1331, USA; (732) 981-0060 / (732) 981-1721 (FAX); http://www.ieee.org/

In addition, many of the documents controlled by the above organizations may also be ordered through a third party:

Global Engineering Documents, 15 Inverness Way, Englewood, CO 80112-5776; (800) 624-3974 / (303) 792-2192; http://www.global.ihs.com/

3 Definitions and notation

3.1 Definitions

3.1.1 Conformance

Several keywords are used to differentiate levels of requirements and optionality, as follows:

3.1.1.1 expected: A keyword used to describe the behavior of the hardware or software in the design models assumed by this specification. Other hardware and software design models may also be implemented.

3.1.1.2 ignored: A keyword that describes bits, bytes, quadlets, octlets or fields whose values are not checked by the recipient.

3.1.1.3 may: A keyword that indicates flexibility of choice with no implied preference.

3.1.1.4 reserved: A keyword used to describe objects (bits, bytes, quadlets, octlets and fields) or the code values assigned to these objects in cases where either the object or the code value is set aside for future standardization. Usage and interpretation may be specified by future extensions to this or other specifications. A reserved object shall be zeroed or, upon development of a future specification, set to a value specified by such a specification. The recipient of a reserved object shall ignore its value. The recipient of an object defined by this specification as other than reserved shall inspect its value and reject reserved code values.

3.1.1.5 shall: A keyword that indicates a mandatory requirement. Designers are required to implement all such mandatory requirements to assure interoperability with other products conforming to this specification.

3.1.1.6 should: A keyword that denotes flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “is recommended.”

3.1.2 Glossary

The following terms are used in this specification:

3.1.2.1 Byte: Eight bits of data, used as a synonym for octet

3.1.2.2 Import/export element: Used to add or remove volumes from the changer device

3.1.2.3 Quadlet: Four bytes of data.

3.1.2.4 Storage element: Where volumes are stored within a changer subunit

3.1.2.5 Transport element: Provides access to volume elements from a player/recorder or similar mechanism

3.1.2.6 Volume: Individual medium, such as a disc or a tape cartridge

3.1.3 Abbreviations

The following are abbreviations that are used in this specification:

AV.C: Audio Video Control

CD: Compact Disc

CD-DA: Compact Disc – Digital Audio

DVD: Digital Versatile Disc

MD: Mini Disc

SACD: Super Audio Compact Disk

3.2 Notation

3.2.1 Numeric values

Decimal and hexadecimal are used within this specification. By editorial convention, decimal numbers are most frequently used to represent quantities or counts. Addresses are uniformly represented by hexadecimal numbers. Hexadecimal numbers are also used when the value represented has an underlying structure that is more apparent in a hexadecimal format than in a decimal format.

Decimal numbers are represented by Arabic numerals without subscripts or by their English names. Hexadecimal numbers are represented by digits from the character set 0 – 9 and A – F followed by the subscript 16. When the subscript is unnecessary to disambiguate the base of the number it may be omitted. For the sake of legibility hexadecimal numbers are separated into groups of four digits separated by spaces.

As an example, 42 and $2A_{16}$ both represent the same numeric value.

3.2.2 Bit, byte and quadlet ordering

This specification uses the facilities of Serial Bus, IEEE 1394, and therefore uses the ordering conventions of Serial Bus in the representation of data structures. In order to promote interoperability with memory buses that may have different ordering conventions, this specification defines the order and significance of bits within bytes, bytes within quadlets and quadlets within octlets in terms of their relative position and not their physically addressed position.

Within a byte, the most significant bit, *msb*, is that which is transmitted first and the least significant bit, *lsb*, is that which is transmitted last on Serial Bus, as illustrated below. The significance of the interior bits uniformly decreases in progression from *msb* to *lsb*.

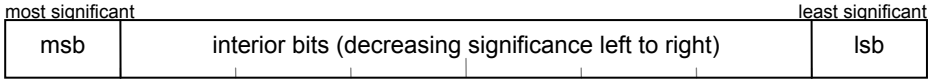

Figure 1 – Bit ordering within a byte

Within a quadlet, the most significant byte is that which is transmitted first and the least significant byte is that which is transmitted last on Serial Bus, as shown below.

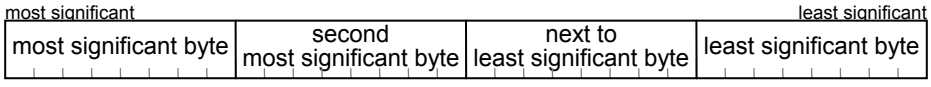

Figure 2 – Byte ordering within a quadlet

Within an octlet, which is frequently used to contain 64-bit Serial Bus addresses, the most significant quadlet is that which is transmitted first and the least significant quadlet is that which is transmitted last on Serial Bus, as the figure below indicates.

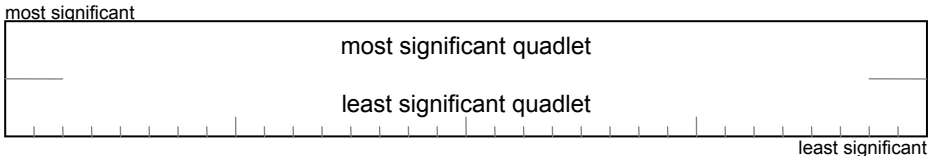

Figure 3 – Quadlet ordering within an octlet

When block transfers take place that are not quadlet aligned or not an integral number of quadlets, no assumptions can be made about the ordering (significance within a quadlet) of bytes at the unaligned beginning or fractional quadlet end of such a block transfer, unless an application has knowledge (outside of the scope of this specification) of the ordering conventions of the other bus.

4 Changer Subunit Model (informative)

4.1 Changer Subunit Model

The changer subunit model consists of:

- volumes
- a storage element containing slots
- transport elements
- an import/export element

This model is illustrated below:

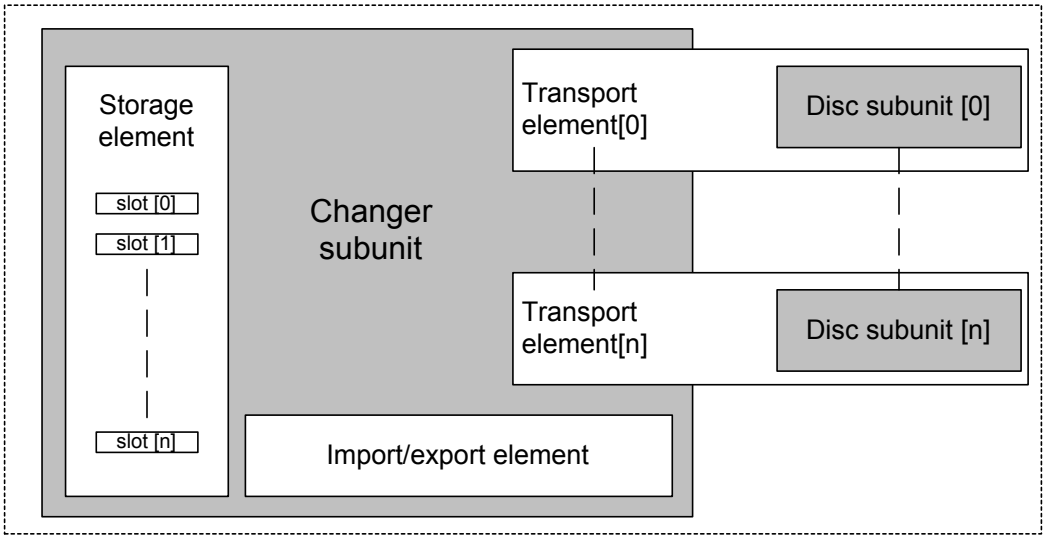

Figure 4 – Changer Subunit Model

4.1.1 Volumes

Volumes are the individual media that are contained within the changer device. The various media types defined in the Disc Subunit Media Specifications [R7][R8][R9][R10] may be handled by the changer subunit. These types include CD, MD, SACD and DVD.

4.1.2 Storage element

Each Changer Subunit includes exactly one storage element: this is where volumes are stored when they are not in a transport element. The storage element has some number of slots in which volume elements can be stored.

4.1.3 Transport elements

Transport elements provide access to volume elements of certain types. Each transport element refers to a disc subunit that is appropriate for all the media types handled by the changer. Future versions of this specification may provide for other types of transport elements, such as tape mechanisms.

4.1.4 Import/Export element

The Import/Export element is used to add or remove volumes from the changer device. A changer subunit may include at most one Import/Export element.

4.1.5 Two Examples (Informative)

Consider a small automotive CD changer, with one drive and space for six discs; CDs are inserted and removed through a dashboard opening. In the AV/C Changer Subunit model, this device would be considered to have up to six volumes (the CDs in the changer), a storage element (the internal space in the changer where the discs are stored), one transport element (the CD drive itself) and an import/export element (the dashboard opening).

A more elaborate example might be a large multi-type changer for a home entertainment system. This changer might include two (or more) drives and storage for hundreds of discs. In such a system, addition and removal of discs by the user might be accomplished by opening a door and adding or removing the media directly. In the AV/C Changer Subunit model this device would be considered to have a large number of volumes (possibly of varying media types, depending on the capabilities of the transports); a single storage element (where all the volumes are stored); two (or more) transport elements; and no import/export element (since the device doesn't "eject" discs).

4.2 Changer subunit operations

There are three main operations performed by the changer subunit:

- moving volumes from a slot in one element to a slot in another element
- allowing a controller to obtain information about the volumes.
- grouping volumes into collections

4.2.1 Volume movement

The basic functionality of the changer subunit is to provide external control over the movement of volumes in changer devices. Within the model of the changer subunit, this volume movement is performed by moving volumes between elements and any slots that the elements may contain.

An example of this functionality is "play disc 1" in a CD jukebox. This would be implemented by a command specifying that the volume described by object ID 1 is to be moved to a transport element, followed by a command to the disc subunit associated with the transport element to play the volume.

As another example, "eject disc 26" would be implemented by specifying that the volume described by object ID 26 is to be moved to the import/export element.

4.2.1.1 Implied movements

The changer subunit must move a volume to its previous location when a new request for another volume to be moved to the current location is made. An example of this is "play disc 1", followed by "play disc 3". The first command will be implemented by requesting the volume described by object ID 1 be moved to a transport element. The second command will be implemented by requesting the volume described by object ID 3 be moved to the transport element. When this second command is issued, the volume currently in the transport element is moved to the storage element.

It is mandatory to implement this implied movement functionality, and not require a controller to request that a volume currently occupying a transport element be moved before allowing another volume to be moved to that transport element.

4.2.2 Access to Volume Information

The changer subunit may provide information about the volumes it stores via various descriptor structures. The structure of the detail for each volume is similar to what would otherwise be provided by a disc subunit for an individual volume.

A changer subunit may handle several hundreds of volumes. Since providing the information in the descriptor structures requires that the volume be read, providing the descriptor information may be a lengthy process. There are two ways in which the descriptor structures can be generated:

- incremental data accumulation
- one-time descriptor building

4.2.2.1 Incremental data accumulation

In the incremental data accumulation way of providing descriptor data, the changer subunit obtains data about the volumes as they are used – i.e. when they are played the changer subunit may store the table of contents and other information that becomes available. Only after all volumes have been used will the descriptor structures be complete for all volumes.

4.2.2.2 One-time descriptor building

The changer subunit may support a descriptor building mode where it will interact with the disc subunits associated with its transport elements to sequentially read all the data from all the volumes it stores. If a changer subunit has more than one transport element, then it may support simultaneously building the descriptor data and another play mode.

4.2.3 Managing volume collections

The changer subunit may allow collections of volumes to be established. Collections can be created according to any criteria, such as artist, genre, etc. These collections, or groups of volumes can be specified as the group of volumes to use for the various play modes that are defined.

If a changer subunit supports collections, then the set of all volume collections is maintained as a root list called the Collection List. Each collection in the Collection List is defined by an entry in that list, including a reference to a child list. Each such child list is called a Contents List.

A collection is referenced through the Object ID of its entry in the Collection Contents List.

5 Changer Subunit Descriptors

5.1 Changer descriptors overview

The structure of the changer subunit descriptors is illustrated below:

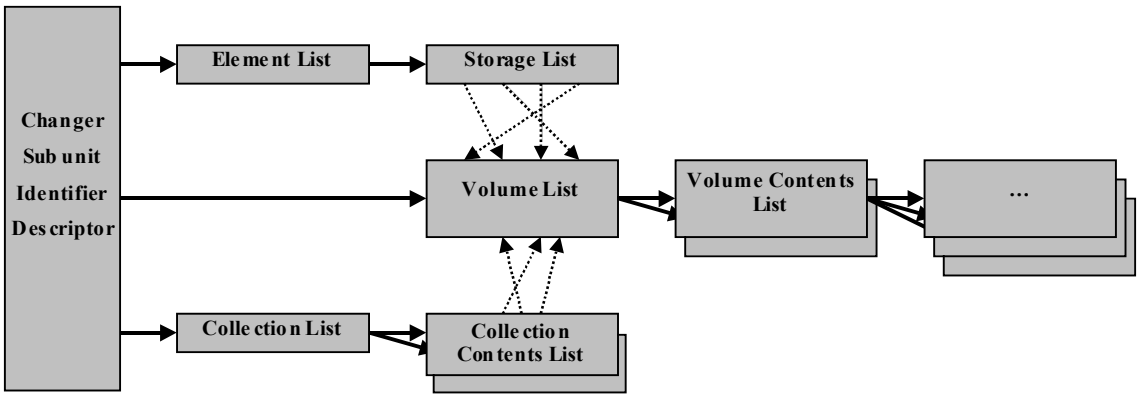

Figure 5 – Changer Subunit Descriptors overview

The changer subunit descriptors contain information that describe the capabilities of the changer subunit as well as the media it contains.

The *changer_subunit_identifier_descriptor* contains the list IDs of the root lists that are present as well as general information about the changer subunit. There are three principal root lists supported by the Changer Subunit model:

Element List: The entries in this mandatory list describe the various elements in the Changer Subunit, including the transport element(s), the storage element, and (if present) the import/export element. Some of the entries in this list will have child lists.

Volume List: This mandatory list includes entries for each volume in the changer. Each of these entries may include additional information about the volumes they describe, starting with a *volume_contents_list*, which is identical in format to a *root_contents_list* (see [R6]).

Collection List: This optional list includes an entry for each collection; each such collection list entry refers to a list of references to the Volume List.

5.2 Changer subunit identifier descriptor

The general structure of a subunit identifier descriptor is defined in [R11]. For convenient reference, an informative illustration of that definition is shown in Figure 6. The definitions of the subunit type-specific fields are given in the following sub-clauses.

Address	Length, bytes	External Read/Write	Contents
$00\ 00_{16}$ $00\ 01_{16}$	2	R	descriptor length
$00\ 02_{16}$	1	R	generation_ID
$00\ 03_{16}$	1	R	size_of_list_ID = S_L
$00\ 04_{16}$	1	R	size_of_object_ID
$00\ 05_{16}$	1	R	size_of_object_position
$00\ 06_{16}$ $00\ 07_{16}$	1	R	number_of_root_lists = n
$00\ 08_{16}$:	S_L	R	root_list_ID[0]
:	:	:	:
: :	S_L	R	root_list_id[n-1]
: :	2	R	subunit_type_dependent_information_length = j
: : :	j	–	subunit_type_dependent_information
: :	2	R	manufacturer_dependent_information_length = k
: : :	k	–	manufacturer_dependent_information
: :	–	–	extended_information (optional)

Figure 6 – Subunit Identifier Descriptor (informative)

5.2.1 *Size_of_list_ID*, *size_of_object_ID*, and *size_of_object_position*

Because the changer subunit may be used with a wide variety of types of disc subunits, this specification *constrains*, rather than defines, the values of these three fields: *size_of_list_ID*, *size_of_object_ID*, and *size_of_object_position*. The values of these fields must satisfy the following rules:

1. The values in these fields must be no less than two (2) and no greater than eight (8).
2. The value of size_of_list_ID must be no greater than the value of size_of_object_ID.
3. The value of size_of_object_ID must be no greater than the value of *size_of_object_position*.

For convenient reference in this document, the values of these three fields are referred to as follows:

S_L for *size_of_list_ID*;

S_O for *size_of_object_ID*; and

S_P for *size_of_object_position*.

Using this notation, the constraints on these fields may be summarized as:

$$2 \le S_L \le S_O \le S_P \le 8$$

5.2.2 *Subunit_type_dependent_information_length* and *subunit_type_dependent_information*

This specification does not define any *subunit_type_dependent_information*. Therefore, a device which conforms to this specification will set *subunit_type_dependent_information_length* to zero.

In the interest of upward compatibility, it is recommended that controllers should allow this field to be non-zero and ignore any *subunit_type_dependent_information* present in this type of descriptor.

6 Changer subunit lists

6.1 Changer subunit lists overview

The changer subunit model defines five list types. In addition, a number of list types defined by the disc subunit model ([R6]) are used in the changer subunit.

Table 1 – List types in the Changer Subunit

List name	list_type	root?	required?	Description & Comments
volume list	$F0_{16}$	y	y	List of all volumes in the changer subunit
element list	$F1_{16}$	y	y	List of all elements in the changer subunit
storage list	$F2_{16}$	n	n	Identifies which volume is in each slot of the storage element
collection list	$F3_{16}$	y	n	List of all collections in the changer subunit
collection contents list	$F4_{16}$	n	–	Required if collection list is present; identifies volumes in a collection
volume contents list	$F5_{16}$	n	n	This list type has the same format as the root contents list in the disc subunit model (see [R6]), except for the *list_type* value, and the fact that this is *not* a root list.

The format of a list descriptor is defined in [R11]; an informative illustration is shown in Figure 7.

Address	Length, bytes	External Read/ Write	Contents
$00\ 00_{16}$ $00\ 01_{16}$	2	R	list_descriptor length
$00\ 02_{16}$	1	R	list_type
:	1	R	attributes
: :	2	R	list_specific_information_length = i
: : :	1	–	list_specific_information
: :	S_P	R	number_of_entry_descriptors = n
: : :	–	–	entry_descriptor[0]
:	:	:	:
: : :	–	–	entry_descriptor[n-1]
: :	–	–	extended_information (optional)

Figure 7 – List descriptor format (informative)

6.2 Volume List

The volume list is a root list. Exactly one volume list must be present in the changer subunit.

The *list_type* of the volume list is $F0_{16}$.

In the *attributes* of the volume list, the *entries_have_object_ID* bit shall be set (one).

The volume list includes no *list_specific_information*, and *list_specific_information_length* shall be zero in the volume list of any device which conforms to this specification.

In the interest of upward compatibility, it is recommended that controllers should allow *list_specific_information_length* to be non-zero and ignore any *list_specific_information* present in this type of descriptor.

The only type of entry allowed in the volume list is Volume Entry, defined in clause 7.1.

6.3 Element List

The element list is a root list. Exactly one element list must be present in the changer subunit.

The *list_type* of the element list is $F1_{16}$.

In the *attributes* of the element list, the *entries_have_object_ID* bit shall be set (one).

The element list includes no *list_specific_information*, and *list_specific_information_length* shall be zero in the volume list of any device which conforms to this specification.

In the interest of upward compatibility, it is recommended that controllers should allow *list_specific_information_length* to be non-zero and ignore any *list_specific_information* present in this type of descriptor.

The only types of entry allowed in the element list are:

- Storage Element Entry, defined in clause 7.2;
- Transport Element Entry, defined in clause 7.3; and
- Import/Export Element Entry, defined in clause 7.4.

6.4 Storage List

The storage list, if supported, is a child of the element list. In particular, it is only allowed to be a child of the storage element entry.

The *list_type* of the storage list is $F2_{16}$.

In the *attributes* of the storage list, the *entries_have_object_ID* bit shall be clear (zero).

The only type of entry allowed in the storage list is Slot Entry, defined in clause 7.5.

6.5 Collection List

The collection list, if supported, is a root list. At most one collection list may be present in the changer subunit.

The *list_type* of the element list is $F3_{16}$.

In the *attributes* of the collection list, the *entries_have_object_ID* bit shall be set (one).

The collection list includes no *list_specific_information*, and *list_specific_information_length* shall be zero in the volume list of any device which conforms to this specification.

In the interest of upward compatibility, it is recommended that controllers should allow *list_specific_information_length* to be non-zero and ignore any *list_specific_information* present in this type of descriptor.

The only type of entry allowed in the collection list is Collection Entry, defined in clause 7.6.

6.6 Collection Contents List

All collection contents lists are child lists of the collection list. If the collection list is supported, then the collection contents list must also be supported; if the collection list is not supported, then the changer subunit shall not support collection contents lists.

The *list_type* of a collection contents list is $F4_{16}$.

In the *attributes* field of a collection contents list, the *entries_have_object_ID* bit shall be clear (zero).

A collection contents list includes no *list_specific_information*, and *list_specific_information_length* shall be zero in every collection contents list of a device which conforms to this specification.

In the interest of upward compatibility, it is recommended that controllers should allow *list_specific_information_length* to be non-zero and ignore any *list_specific_information* present in this type of descriptor.

The only type of entry allowed in a collection contents list is Collection Contents Entry, defined in clause 7.7.

6.7 Volume Contents List

The format of a volume contents list is the same as that of a root contents list as defined in [R6]. The only modifications of that definition for volume contents lists in the Changer Subunit are these:

- The *list_type* of a volume contents list is $F5_{16}$.
- All volume contents lists are child lists.
- A volume contents list may include a collection reference information block (see [R5]) as defined in clause 6.7.1.

6.7.1 Collection_reference_info_block

The *collection_reference_info_block* is defined below:

Address Offset	Msb							Lsb
00_{16}	Compound_length							
01_{16}	Info_block_type = 89 01_{16}							
02_{16}	Primary_fields_length							
03_{16}								
04_{16}	Number_of_collections (n)							
05_{16}	Collection_Identifier[0]							
06_{16}								
:	:							
	Collection_Identifier[n-1]							

Figure 8 – Collection_reference_info_block

The *compound_length* field specifies the number of bytes for the remainder of this information block (including any nested info blocks which may occur after the last non-info block field). Note that currently there are no nested information blocks shown for this structure, but controllers should be prepared for any blocks to be found while parsing this structure.

The *primary_fields_length* specifies the number of bytes for the remaining non-info block fields of this structure. All nested info blocks shall appear after this.

The *number_of_collections* field specifies how many *collection_identifier* entries are following. This number should always be the maximum number of collections that the volume can be associated with – the *collection_identifier* entries can indicate a null value.

Each *collection_identifier* field specifies the identifier of the *collection_contents_list* to which the volume belongs. The value 00 00_{16} means that the *collection_identifier* entry does not specify an identifier of a *collection_contents_list*.

7 Changer Subunit Entries

The changer subunit defines seven entry types, as listed in Table 2

Table 2 – Entry types in the Changer Subunit

List name	entry_type	Description & Comments
volume entry	$F0_{16}$	Describes one volume in the changer subunit
storage element entry	$F1_{16}$	Describes the storage element in the changer subunit
transport element entry	$F2_{16}$	Describes a transport element in the changer subunit
import/export element entry	$F3_{16}$	Describes the import/export element in the changer subunit
slot entry	$F4_{16}$	Identifies the volume in one slot of the storage element
collection entry	$F5_{16}$	Describes one collection in the changer subunit
collection contents entry	$F6_{16}$	Identifies one volume in a collection

The format of an entry descriptor is defined in [R11]; an informative illustration is shown in Figure 9.

Address	Length, bytes	External Read/Write	Contents
$00\ 00_{16}$ $00\ 01_{16}$	2	R	entry_descriptor length
$00\ 02_{16}$	1	R	entry_type
:	1	R	attributes
: :	S_L	R	child_list_id (optional)
: : :	S_O	–	object_ID (optional)
: :	2	R	entry_specific_information_length = i
: : :	i	–	entry_specific_information
: :	–	–	extended_information (optional)

Figure 9 – Entry descriptor format (informative)

7.1 Volume Entry

The value in the *entry_type* field of a volume entry is $F0_{16}$.

In the *attributes* field of a volume entry, the *has_child_ID* bit may be set (one) or clear (zero):

If *has_child_ID* is set (one) then the *child_list_ID* field refers to the root contents list of the volume.

If *has_child_ID* is clear (zero) then (in accordance with [R11]) the *child_list_ID* field is absent from the descriptor.

The *entry_specific_information* field for a volume entry follows the following format:

Offset (within field)	Length, bytes	External Read/Write	Contents
$00\ 00_{16}$:	S_O	R	element_ID
: :	S_P	R	slot_number

Figure 10 – Volume Entry *entry_specific_information* field format

The *element_ID* field contains the object ID of the Changer Subunit element where the volume is currently located.

If *element_ID* refers to the storage element, the *slot_number* field contains either $FF…FF_{16}$, or it indicates the position of the volume in the storage element. If *element_ID* does not refer to the storage element, then *slot_number* contains $FF…FF_{16}$. It is recommended that controllers ignore the contents of *slot_number* in cases where *element_ID* does not refer to the storage element, as the use of this field may change in future versions of this specification.

7.2 Storage Element Entry

The value in the *entry_type* field of a storage element entry is $F1_{16}$.

In the *attributes* field of a storage element entry, the *has_child_ID* bit may be set (one) or clear (zero).

If *has_child_ID* is set (one) then the *child_list_ID* field refers to a storage list: the entries in that list specify which volumes occupy the various slots in the storage element.

If *has_child_ID* is clear (zero) then (in accordance with [R11]) the *child_list_ID* field is absent from the descriptor.

The *entry_specific_information* field for a storage element entry follows the following format:

Offset (within field)	Length, bytes	External Read/Write	Contents
$00\ 00_{16}$:	S_P	R	number_of_slots

Figure 11 – Storage Element Entry *entry_specific_information* field format

The *number_of_slots* field specifies how many slots are in the storage element.

7.3 Transport Element Entry

The value in the *entry_type* field of a transport element entry is $F2_{16}$.

In the *attributes* field of a transport element entry, the *has_child_ID* bit shall be clear (zero).

The *entry_specific_information* field for a transport element entry follows the following format:

Offset (within field)	Length, bytes	External Read/Write	Contents
$00\ 00_{16}$:	–	R	associated_subunit_identifier

Figure 12 – Transport Element Entry *entry_specific_information* field format

The *associated_subunit_identifier* field identifies the subunit for this transport element. This field has the same format as the "AV/C address" portion of an AV/C command frame, including *subunit_type* and *subunit_ID* sub-fields, and possibly additional "extended" bytes: see [R4] for more information.

7.4 Import/Export Element Entry

The value in the *entry_type* field of an import/export element entry is $F3_{16}$.

In the *attributes* field of an import/export element entry, the *has_child_ID* bit shall be clear (zero).

The *entry_specific_information* field for an import/export element entry shall be empty.

7.5 Slot Entry

The value in the *entry_type* field of a slot entry is $F4_{16}$.

In the *attributes* field of a slot entry, the *has_child_ID* bit shall be clear (zero).

The *entry_specific_information* field for a slot entry follows the following format:

Offset (within field)	Length, bytes	External Read/Write	Contents
$00\ 00_{16}$:	S_O	R	volume_ID
: :	S_P	R	slot_number (optional)

Figure 13 – Slot Entry *entry_specific_information* field format

The *volume_ID* field identifies which volume occupies the slot associated with this entry descriptor. The value of this field is equal to the *object_ID* field in the volume entry descriptor for the volume in the volume list.

The *slot_number* field, if present, specifies which slot in the storage element is associated with this entry. If the *slot_number* field is absent, then the slot number is inferred from context:

If this is the first entry in the storage list, then the slot number is taken to be zero (0).

Otherwise, the slot number is taken to be one greater than the slot number of the previous entry in the storage list

If the *slot_number* field is present, the value it contains shall be no less than the value that would be inferred had the field been absent.

7.6 Collection Entry

The value in the *entry_type* field of a collection entry is $F5_{16}$.

In the *attributes* field of a collection entry, the *has_child_ID* bit shall be set (one), and the *child_list_ID* field shall contain the list ID of a Collection Contents List.

The *entry_specific_information* field contains no non-info block fields. The *entry_specific_information* may be empty, or it contain a single name info block, as defined in [R5].

7.7 Collection Contents Entry

The value in the *entry_type* field of a collection contents entry is $F6_{16}$.

In the *attributes* field of a collection contents entry, the *has_child_ID* bit shall be clear (zero).

The *entry_specific_information* field for a collection contents entry follows the following format:

Offset (within field)	Length, bytes	External Read/Write	Contents
$00\ 00_{16}$:	S_O	R	volume_ID

Figure 14 – Collection Contents Entry *entry_specific_information* field format

The *volume_ID* field identifies the volume associated with this entry descriptor. The value of this field is equal to the *object_ID* field in the volume entry descriptor for the volume in the volume list.

8 Changer Subunit Commands

8.1 Overview

One command is defined for the changer subunit:

Table 3 – Support level of changer subunits commands

Opcode	Value	Support level (by ctype)			Comments
		C	S	N	
MOVE VOLUME	45_{16}	M	-	-	Move volumes between elements

8.2 MOVE VOLUME command

8.2.1 MOVE VOLUME control command

The MOVE VOLUME command is used to move volumes between elements in the changer subunit. The format of the command is defined in Figure 15 below:

	length	msb							Lsb
opcode	1	MOVE VOLUME (45_{16})							
operand[0]	S_O	source_element_or_volume_ID							
:									
:	S_O	destination_element_ID							
:									

Figure 15 – MOVE VOLUME control command

The *source_element_or_volume_ID* field contains an object ID: either of the import/export or transport element from which the volume is to be moved, or of the volume itself. Note that this may not be the object ID of the storage element.

The *destination_element_ID* field contains the object ID of the element to which the volume is to be moved. If this is a transport element, then the volume is moved to that element; if this is the import/export element, then the volume is ejected from the changer; if this is the storage element, then the volume is moved into storage and the storage list (if supported) is updated accordingly.

8.2.2 MOVE VOLUME control response

If the changer subunit is unable to perform the requested movement, then the changer subunit shall return a REJECTED response.

If the *source_element_or_volume_ID* field specifies the import/export element or a transport element, and if no volume is available at the specified element, then the changer subunit shall return a REJECTED response.

Any MOVE VOLUME control command may elicit an INTERIM response. If the physical operation of moving volumes between elements might take longer than 100ms, the target shall return an INTERIM response, followed by a final response.

It is important that when the MOVE VOLUME command is completed that a controller be able to then interact with the disc subunit(s) associated with the changer subunit's transport elements.

In an INTERIM or ACCEPTED response frame, all fields shall be the same as the control command, with the following exceptions:

If, in the command frame, the *source_element_or_volume_ID* specified an element ID, then in an INTERIM response frame that field may be set to the volume ID of the volume moved.

If, in the command frame, the *source_element_or_volume_ID* specified an element ID, then in an ACCEPTED response frame that field shall be set to the volume ID of the volume moved.

8.2.2.1 INTERIM response rule

This INTERIM response rule shall be observed when an INTERIM response is used.

Since another MOVE VOLUME control command may be received before the final response is returned, the following implementation rule shall be observed:

If the target receives the MOVE VOLUME control command before a final response is sent to an outstanding MOVE VOLUME command, then the target shall immediately complete the oustanding response with REJECTED, and return an INTERIM response for the new control command. This rule is illustrated below:

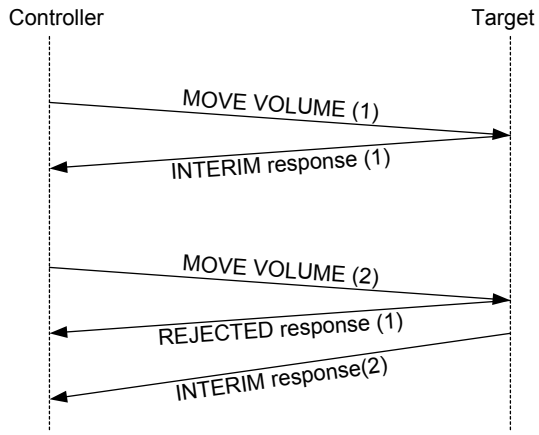

Figure 16 – INTERIM response rule

It is necessary for controller implementations to distinguish responses that are received by examining the source and destination fields within the commands and responses.

Annex A.
(Informative)

Compliance

A.1. Mandatory and Optional Features

A.1.1 Changer subunit lists

Changer subunit list types are defined in Table 1. The requirements for changer subunit lists are given in clause 6.

The mandatory and optional support for changer subunit lists is summarized as follows:-

- A changer target supports one volume (root) list and one element (root) list.
- A changer target may support one storage list for each entry in the element list.
- A changer target may support one collection (root) list.
- A changer target supports one collection contents list for each entry in the collection list (if present).
- A changer target may support one volume contents list for each entry in the volume list.

A.1.2 Changer subunit commands

There is only one subunit specific command – MOVE VOLUME – which is defined in section 8.2. A changer subunit target supports the MOVE VOLUME command.

A changer subunit target supports the following commands defined in [R11] to allow a controller to access the AV/C descriptors describing the facilities, contents and state of the changer:-

- OPEN DESCRIPTOR control command, subfunction Read open
- OPEN DESCRIPTOR control command, subfunction Close
- READ DESCRIPTOR control command

A changer subunit target may support the following commands defined in [R11] to allow a controller to access the AV/C descriptors describing the facilities, contents and state of the changer:-

- OPEN DESCRIPTOR status command
- OPEN DESCRIPTOR notify command

A.2. Test scenarios

A changer target should be tested under the control of a single controller and under the control of two controllers.

The tests should include enumeration of all elements in a changer target, enumeration of all volumes (and their contents where supported), and at least some volume movements between all elements pairwise. Verification of

movement to a transport element should be performed by appropriate commands given to the subunit attached to the transport element.

Annex B.
(Informative)

Assignment of Volume Object IDs

B.1. Overview

This specification does not define how a changer subunit should assign IDs to volumes. The only requirement for volume IDs made by this specification is that, while a volume is in the changer, its ID must not change. If the user opens the changer and rearranges the volumes stored there, the assignment of IDs to volumes may change completely: old volume IDs may not be valid, or they may still be valid but referring to different volumes, or they may still be valid and refer to the same volume.

Also, different changer subunits need not assign the same ID to the same volume. Indeed, a changer subunit may assign one ID to a volume on one occasion; and on another occasion, when the same volume is put into the changer, the device may assign an entirely different ID.

This flexibility has been included deliberately, to accommodate changers with different capabilities. To illustrate some of the options for implementations of the changer subunit as defined in this document, the following subsections present two examples of different types of changers with very different capabilities.

B.2. Example: A Carousel-Type Changer

One popular type of CD changer is the carousel-type changer. These changers have a large platter with five or six recesses: each recess can hold a single disc. The changer has a single transport in one of the media positions: to play a disc, the changer rotates the carousel until the recess holding the disc in question is aligned under the transport; then the disc is lifted off the carousel and played. When a new disc is to be played, the old disc is lowered back into its recess in the carousel, the carousel rotates to the new position, and the new disc is raised to the transport. (The tray for such a changer is illustrated in Figure B-1.)

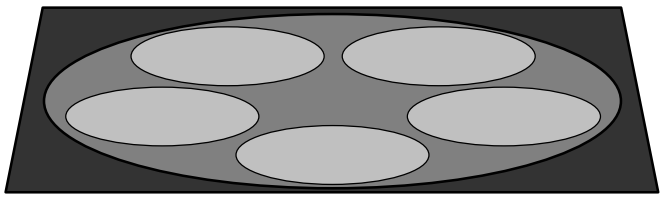

Figure B1 – Tray of a 5-position carousel-type changer.

Such a changer may not have enough capabilities to read a unique identifier from the volumes on the carousel; and changers of this type may well be unable to move volumes between recesses (or “slots” in the terminology of this specification). However, changers of this type are typically very able to determine the rotational position of the carousel, and which recess (“slot”) is currently below the transport.

For changers of this type, subunit could use the slot numbers as volume object IDs. Since the changer cannot move volumes from one “slot” to another, this ID will remain uniquely associated with a particular disc (or with no disc!) as long as the user doesn’t rearrange the discs.

B.3. Example: A Jukebox-Type Changer

Some more advanced changers available today are based on a more sophisticated mechanism, including an array of slots into which discs may be placed, and an arm which can move discs arbitrarily between slots. A changer of this type might be able to read a unique identifier from a disc, identify which (if any) of the discs were changed when the door was opened, and be able to detect which slots are occupied and which are not.

For the purposes of discussion, consider a changer with the following characteristics and capabilities:

- 100 slots;
- 2 transports;
- Able to detect which slots are full;
- Able to detect when any disc is changed;
- Able to read a media-identifying number from each volume.

Such a changer might use the AV/C Disc and Changer subunits as follows:

- Two disc subunits, one for each transport;
- One changer subunit, with two transport elements (one per disc subunit) and a storage element;
- Depending on how discs are inserted into and removed from the changer, the changer might also include an import/export element.

A changer of this type might use a simple cyclic scheme for assigning object IDs to volumes. For example, suppose the changer uses 6-byte object IDs; and suppose the first 2^{32} object IDs (from 000000000000_{16} to $0000FFFFFFFF_{16}$ inclusive) are reserved for objects other than volumes. The changer would arbitrarily assign object ID 000100000000_{16} to the first disc in the changer, 000100000001_{16} to the second, etc.; and after finally assigning object ID $FFFFFFFFFFFF_{16}$, the changer would begin again at 000100000000_{16}.

The only major issue in this scheme is volume ID roll-over: if two objects are assigned the same ID, a controller could become confused. This problem can be overcome with the use of sufficiently large object IDs. In the present example (100-slot changer, 6-byte object IDs, 2^{32} IDs reserved for other purposes), suppose all 100 discs in the device were changed once per second: then it would be more than 89000 years before an object ID were re-used. It is unlikely that a changer would remain in service for that long.

There are, of course other schemes that could be used to satisfy the requirements. These are just two possible implementations.

Annex C.
(Informative)

Some Controller-Changer Protocols

C.1. Overview

This annex gives some illustrative example, showing how a controller and changer can cooperate to perform some common tasks. For the purposes of this annex, we assume the changer has the following characteristics:

- One Disc subunit;
- One Changer Subunit, with one storage element and one transport element, associated with the disc subunit;
- The volumes are all CD-DA discs.

C.1.1 Example: Collecting a Volume List

The controller uses the following commands, all addressed to the subunit:-

1) OPEN DESCRIPTOR/Read Open, descriptor specifier type 00 (subunit identifier descriptor)

2) READ DESCRIPTOR descriptor specifier type 00 (subunit identifier descriptor)

3) OPEN DESCRIPTOR/Close, descriptor specifier type 00 (subunit identifier descriptor)

4) Search the root lists in the subunit descriptor for the volume list:-

   a) OPEN DESCRIPTOR/Read Open, descriptor specifier type 10 (list descriptor by list_ID), root_list_ID[0] (from the subunit descriptor)

   b) READ DESCRIPTOR descriptor specifier type 10 (list descriptor by list_ID), root_list_ID[0] (from the subunit descriptor)

   c) OPEN DESCRIPTOR/Close descriptor specifier type 10 (list descriptor by list_ID), root_list_ID[0] (from the subunit descriptor)

   d) If the list_type is not F0 (volume list), then repeat the previous three steps for the next root_list_ID.

5) Read the volume entry information from each entry_descriptor in the volume list (alll entry descriptors will be of type Volume Entry). If the *has_child_ID* flag is set, then read the descripitor with the child_list_ID to access the volume contents list:-

   a) OPEN DESCRIPTOR/Read Open, descriptor specifier type 10 (list descriptor by list_ID), child_list_ID (from the entry_descriptor)

   b) READ DESCRIPTOR descriptor specifier type 10 (list descriptor by list_ID), child_list_ID (from the entry_descriptor)

   c) OPEN DESCRIPTOR/Close, descriptor specifier type 10 (list descriptor by list_ID), child_list_ID (from the entry_descriptor)

C.1.2 Example: Single Volume Selection and Play-Back

One common task performed by a controller and a changer is the selection and play-back of a single volume. We assume:

- the object ID of the volume to be played it 42_{16};
- the changer has one transport element (number 0);
- the volumes are CDs;
- the plug and stream configurations for playback are already established.

To play the volume, the controller:

1) uses a MOVE VOLUME command to the Changer Subunit to move volume 42_{16} to transport element 0; and

2) uses a PLAY command to the Disc element to start playback of the disc.

C.1.3 Example: Performance List Composition and Execution

A particular issue is the support of performance lists in a changer. Since performance list are stored in the disc subunit, this means that a performance list has to refer to a track on a volume that is not currently loaded into the disc subunit (the volume is in the changer subunit). This is enabled by the use of descriptors in the disc subunit that refer to the changer subunit. This allows performance lists such as: "track 1 on disc 4; then track 7 on disc 93; then ...."

A performance list is defined as a set of object references. An enhancement to AV/C Descriptors [R11] allows descriptors to refer to objects by *subunit_specifier*, *root_list_ID*, *list_type* and *object_ID*, or to refer to objects by *subunit_specifier* and *object_ID*. Thus a disc performance list can be created that uses these descriptors to refer to entries in the collection list of the changer subunit.

The changer would typically access the volume information, as described in C.1.1 above, interact with the user to allow the user to identify the volumes and tracks to be included in a performance list, and then build the desired performance list(s) in the disc subunit as described in the appropriate specification (see [R6] and [R7], [R8], [R9] or [R10] as appropriate). Each entry in the performance list would include a subunit identifier to cross reference to the changer.

NOTE – At the time of preparation of this specification, an enhancement is anticipated to the AV/C disc subunit specification to support the creation of performance lists with descriptors referencing the individual objects using Entry descriptors specified by subunit_specifier, root_list_ID, list_type and object_ID, or specified by subunit_identifier and object_ID.